Programación de Microservicios con Spring Boot y Red Hat SSO

Nivel Avanzado

Instructor: Carlos Carreño
ccarrenovi@gmail.com

Modulo 3. Comunicación entre Microservicios

Objetivo: Aprender a desarrollar microservicios con Spring Boot 3, incluyendo configuración, comunicación entre servicios, seguridad y despliegue

Duración: 1h

Comunicación entre Microservicios

- **La interacción entre un servicio local y un *microservicio externo*** en Spring Boot, implica manejar las **comunicaciones entre microservicios**, generalmente vía HTTP, aunque también se puede hacer por mensajería (Kafka, RabbitMQ).

Ejemplo: Servicio Pedidos

Escenario:

- Un servicio de **Pedidos** (PedidoService) en tu app actual.
- Un microservicio de Usuarios, con una API REST disponible en:

http://usuarios-service/api/usuarios/{id}

- El servicio de **Pedidos (**PedidoService**)** necesita obtener los datos del **Usuario** antes de crear un pedido.

RestTemplate vs WebClient

- Usar **RestTemplate** o **WebClient**

```java
@Service
public class UsuarioClient {

    private final RestTemplate restTemplate;

    public UsuarioClient(RestTemplateBuilder builder) {
        this.restTemplate = builder.build();
    }

    public UsuarioDTO obtenerUsuarioPorId(Long id) {
        String url = "http://usuarios-service/api/usuarios/" + id;
        return restTemplate.getForObject(url, UsuarioDTO.class);
    }
}
```

continuación

- Registra el RestTemplate como @Bean (si no usas Spring Boot 3):

```
@Configuration
public class AppConfig {
    @Bean
    public RestTemplate restTemplate() {
        return new RestTemplate();
    }
}
```

WebClient (Es mas reactivo)

```java
@Service
public class UsuarioClient {

    private final WebClient webClient = WebClient.create("http://usuarios-service/api");

    public UsuarioDTO obtenerUsuarioPorId(Long id) {
        return webClient.get()
                .uri("/usuarios/{id}", id)
                .retrieve()
                .bodyToMono(UsuarioDTO.class)
                .block(); // o usar de forma reactiva
    }
}
```

Ejemplo: PedidoService

```java
@Service
public class PedidoService {

    private final UsuarioClient usuarioClient;

    @Autowired
    public PedidoService(UsuarioClient usuarioClient) {
        this.usuarioClient = usuarioClient;
    }

    public Pedido crearPedido(Long usuarioId, Pedido datosPedido) {
        UsuarioDTO usuario = usuarioClient.obtenerUsuarioPorId(usuarioId);

        if (usuario == null) {
            throw new RuntimeException("Usuario no encontrado");
        }

        // lógica para guardar pedido, asignar usuario, etc.
        datosPedido.setUsuarioId(usuarioId);
        // pedidoRepository.save(datosPedido);

        return datosPedido;
    }
}
```

Introducción a OpenFeign

OpenFeign es un cliente HTTP declarativo usado en microservicios para comunicarse fácilmente con otros servicios REST.

⚠ Problemas típicos sin resiliencia

- ❑Llamadas bloqueadas o lentas a otros microservicios
- ❑Sin control de timeout: riesgo de espera infinita
- ❑Sin fallback: cuando el servicio externo falla, toda la cadena se rompe

Dependencia de OpenFeign

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-openfeign</artifactId>
</dependency>
<dependency>
    <groupId>io.github.resilience4j</groupId>
    <artifactId>resilience4j-spring-boot3</artifactId>
</dependency>
```

OpenFeign Client

- Spring Cloud ofrece una alternativa muy elegante, usar OpenFeign

```
@FeignClient(name = "usuarios-service")
public interface UsuarioFeignClient {
    @GetMapping("/api/usuarios/{id}")
    UsuarioDTO obtenerUsuario(@PathVariable("id") Long id);
}
```

- Luego se inyecta directamente como cualquier servicio

Manejo de tiempo de espera y resiliencia básica

- Implementación de Fallback

```java
import org.springframework.stereotype.Component;

@Component
public class ServicioExternoFallback implements ServicioExternoClient {

    @Override
    public String obtenerDatos() {
        return "Servicio externo no disponible. Respuesta de fallback.";
    }
}
```

continuación

- Configuración de Timeout (application.properties o application.yml)

```
feign.client.config.default.connectTimeout=2000   # 2 segundos de conexión
feign.client.config.default.readTimeout=3000      # 3 segundos de espera de respuesta
```

Lab

- Ejercicio práctico: Consumiendo otro Microservicio